U T E M
Facultad de Ingeniería
Departamento de Informática y Computación

Programación INF627
2010

Recursividad

La recursión es un concepto amplio, difícil de precisar. Aparece en numerosas actividades de la vida diaria, por ejemplo, en una fotografía de una fotografía. Otro caso muy ilustrativo de recursión es el que se presenta en los programas de televisión en los cuales un periodista transfiere el control a otro periodista que se encuentra en otra ciudad, y éste hace lo propio con un tercero. Aquí nos limitaremos a estudiar la recursividad desde el punto de vista de programación.

La recursión permite definir un objeto (problemas, estructuras de datos) en términos de sí mismo. Casos típicos de estructuras de datos definidas de manera recursiva son los árboles y las listas ligadas. Algunos ejemplos de problemas que se definen recursivamente son el factorial de un número, la serie de Fibonacci, etc.

Ejemplo:
Para todo número natural **n**, se llama **n factorial** o **factorial de n** al producto de todos los naturales desde 1 hasta **n**:

$$n! = 1 \times 2 \times 3 \times 4 \times \ldots \times (n-1) \times n$$

Se define 0! = 1, para que la relación $n! = n \times (n-1)!$ sea también válida para $n = 1$. Esta relación permite definir los factoriales por recursividad.

$$n! = \begin{cases} \text{si } n = 0 & \Rightarrow 1 \\ \text{si } n \geqslant 1 & \Rightarrow (n-1)! \cdot n \end{cases}$$

Implementación de la función factorial:

```
int factorial(int n){
  if(n==0)
     return 1;
  else
     return factorial(n-1)*n;
}
```

Observación: la función comienza consultando si n = 0, lo que equivale a la condición de finalización de la función.

Implementación de procedimiento (función void) de factorial:

```
void factor(int n,int *res){
  int aux;
  if(n==0)
     *res = 1;
  else{
     factor(n-1,&aux);
```

```
      *res = aux * n;
   }
}
```

Observación: El parámetro de entrada n, indica el valor que se calculará, mientras que el parámetro de entrada/salida res indica la respuesta que nos entregará la función.

Implementación de programa principal:

```
int main()
{
  int n, solucion;
  printf("Ingrese numero : ");
  scanf("%d",&n);

  printf("El factorial de %d es %d\n",n,factorial(n));
  factor(n,&solucion);
  printf("El factorial de %d es %d\n",n,solucion);

  system("pause");
}
```

Observación: Se destaca la invocación de la función factorial (que es parte de una sentencia), y la llamada del procedimiento factor (que es una sentencia), que tiene un parámetro por valor y otro por referencia.

Recursividad sobre arreglos:

```
void Mostrar(int a[],int n)
{
    if(n==0)
    {  // no hace nada
    }
    else
    {
       MostrarR(a,n-1);
       printf("%d\n",a[n-1]);
    }
}
```

Observación: Se llama a la función con el arreglo y la cantidad de elementos (n). Si el arreglo tiene 0 elementos no hace nada. La llamada recursiva se hace al llamar a la misma función con el arreglo y la cantidad de elementos disminuida en uno.

Ejercicios Propuestos

1. Escribir la función recursiva **Combinatoria(n, m)**, que permite retornar combinatoria de n sobre m.
2. Escribir la función recursiva **Euclides(n, m)**, que permite retornar el máximo común divisor (MCD) entre n y m utilizando el algoritmo de Euclides.
3. Escribir la función recursiva **Esta(A, n, ele)**, que permite determinar si el elemento ele se encuentra o no en el arreglo A. Debe retornara 1 si la búsqueda fue exitosa y 0 en caso contrario.
4. Escribir la función recursiva **ContarPares(A, n)**, que permite contabilizar cuántos elementos pares contiene el arreglo A.
5. Escribir la función recursiva **posMayorDigito(A, n)**, que permite retornar la posición del número que posea la mayor suma de dígitos al interior del arreglo A.